



Tracing and trace analysis strategies for GPU-accelerated HSA programs

Progress Report Meeting
December 7, 2017

Paul Margheritta Michel Dagenais

DORSAL lab
École Polytechnique de Montréal

Introduction

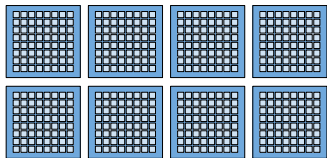


GPU: Graphics Processing Unit

SIMD-based highly parallel architecture (up to several thousand processing elements)

Purpose: **graphics** (video games, etc.)
vs **GPGPU** (computation, deep learning, etc.)

Increasingly **popular**, **powerful** and more easily **programmable**



(8 compute units,
8 64 processing elements)

Research goals

Explore current **tracing and profiling tools** for GPU-accelerated programs

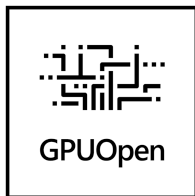
Provide **tracing mechanisms** in a GPU compute-oriented runtime

Create **post-tracing processing features** for our traces

Design **views** for better understanding



Software context: GPU-related tools



HSA: an architecture that speeds up communication between devices in a heterogenous context

ROCm: a HSA-based GPU runtime that we can use to run compute kernels

CLOC: a tool to generate HSA code objects from OpenCL kernels

CodeXL: an open-source debugging and performance analysis tool for HSA and OpenCL



Software context: open-source analysis tools



LTTng: helps us trace events in the ROCr runtime

Babeltrace

Babeltrace: helps us visualize trace and create post-tracing processing scripts



Trace Compass: helps us create views for our traces

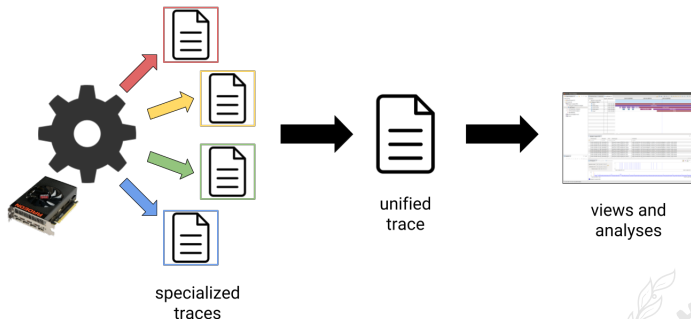


General concept of LTTng-HSA

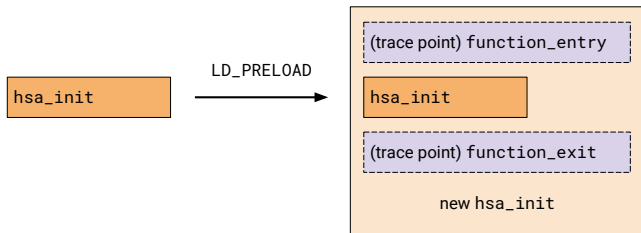
Our focus: tracing **GPU-related CPU events**

The LTTng instrumentation is inserted with a collection of **preloaded libraries** that intercept relevant functions

Not all events can be traced in one execution: we **trace separately and merge** the resulting traces



Synchronous tracing targets



Call stack target: all HSA API functions instrumented at entry and exit

Queue pro ling target: traces the state of user-mode queues and the enqueueing of GPU kernel dispatch packets



Asynchronous tracing targets

The events from these tracing targets require **sorting** when merged into a larger trace.

Kernel timing target: uses a specific type of queue to record GPU kernel start/end times

Performance counters target: uses the SoftCP mode to define pre- and post-dispatch callbacks that set up mechanisms from GPUPerfAPI to gather GPU counters.

Some useful counters:

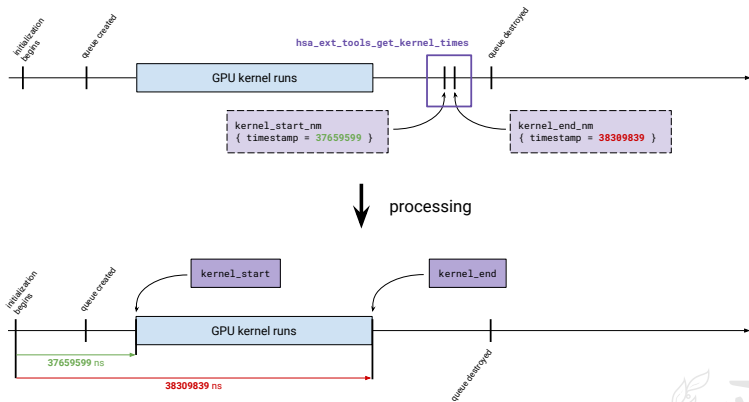
CacheHit: the ratio of GPU L2 cache hits

VALUInsts: the average number of vector ALU instructions executed per work item



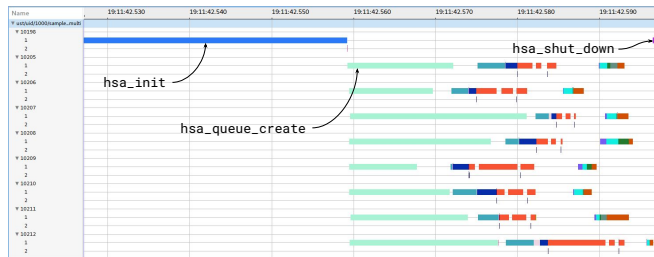
Trace merging and event sorting

Traces are **merged** with Babeltrace then asynchronous events are **sorted** in the larger trace:

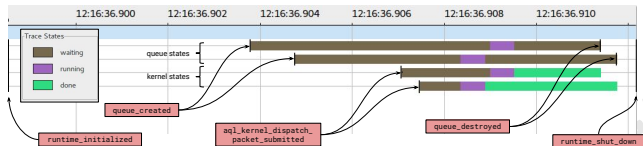


Trace Compass views

Call stack view:



Queue profiling view:



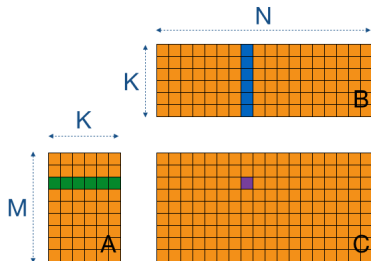
Experimental results

Context: a GPU-accelerated matrix multiplication algorithm

We run our tests on ROCr/HSA with OpenCL kernels compiled with CLOC

3 versions of the algorithm are compared:

- 1 naive algorithm with pseudo-random accesses
- 2 naive algorithm with accesses in the right order
- 3 more optimized algorithm with tiling



(image by Cedric Nugteren)



Experimental results

Relevant information is provided by the **kernel timing target** and the **performance counters target** to gradually improve the algorithm

Version 3 is **faster** than version 2, which is **faster** than version 1

Performance counters show that:

- version 1 has a high **L2 cache miss ratio**
- version 1 and 2 have a high **number of vector and scalar instructions**



Additional results and contributions

Created an LTTng kernel module to trace events from the **AMD Linux Kernel drivers**

Analyzed the **overhead** of our solution

Automated the generation of **interception mechanisms** for the call stack target

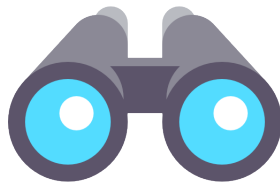


Possible improvements

Improve the reliability of **trace merging and event sorting**

Provide tracing targets for **specific runtimes** (OpenCL, OpenGL, deep learning frameworks, etc.)

Go deeper in the **Linux kernel-side** analysis



Thank you!
Any questions?

`paul.margheritta@polymtl.ca`
`github.com/pmargheritta`

